1986

A decision support system for the diagnosis of aircraft emergencies

Porter, Olen D.

Monterey, California: U.S. Naval Postgraduate School

http://hdl.handle.net/10945/21761

NPS52-86-027

NAVAL POSTGRADUATE SCHOOL

Monterey, California

P7424

THESIS

A DECISION SUPPORT SYSTEM FOR
THE DIAGNOSIS OF AIRCRAFT EMERGENCIES

Olen D. Porter

December 1986

Thesis Advisor: Neil C. Rowe

Approved for public release; distribution unlimited

Prepared for:
Office of Naval Research
Arlington, VA 22217

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-86-027	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A DECISION SUPPORT SYSTEM FOR THE DIAGNOSIS OF AIRCRAFT EMERGENCIES		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) OLEN D. PORTER		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS MIPR ATEC 88-86
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Combat Development Experimentation Center Fort Ord, CA 93941-5012		12. REPORT DATE December 1986
		13. NUMBER OF PAGES 89
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

EXPERT SYSTEMS
AIRCRAFT PILOTING
PILOT'S ASSOCIATE SYSTEMS
MEANS-ENDS ANALYSIS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of this research is to show the feasibility of an expert system that utilizes the existing sensors aboard an aircraft to aid the pilot in the diagnosis of single and compound emergencies. A passive expert planner is proposed that utilizes multiple and domain dependent knowledge-bases. The system is implemented on a personal computer, using the USMC AH-1T attack helicopter as a modeling platform. An effort is made to quantify the amount of information processing necessary to adequately define emergencies. Performance of the system was also evaluated.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

The purpose of this research is to show the feasibility of an expert system that utilizes the existing sensors aboard an aircraft to aid the pilot in the diagnosis of single and compound emergencies. A passive expert planner is proposed that utilizes multiple and domain dependent knowledge-bases. The system is implemented on a personal computer, using the USMC AH-1T attack helicopter as a modeling platform. An effort is made to quantify the amount of information processing necessary to adequately define emergencies. Performance of the system was also evaluated.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Future systems will provide the pilot with information rather than raw data. This information will probably be presented to the pilot in the form of situation reporting, presentation of the options, and probabilities connected with various courses of action [Ref. 1: p. 28]. James A. Guffy, unit chief, Advanced AI Technology Concepts, stated [Ref. 2: p.66]:

> The way it stands now, a pilot is often drowning in data but is starved for information, That's the problem the Pilot's Associate program was created to address. Its role is to increase the pilot's decision-making capability and improve mission effectiveness.

The need for an improved decision-making capability is evident from aircraft accident statistics. The cause/factor elements involved in aircraft accidents may be grouped into three general categories:

(1) Environmental extreme.
(2) Material failure.
(3) Human error.

Environmental extremes are usually external to the cockpit and beyond the pilot's control. Material failures are hardware malfunctions and structural failures. Human errors are procedural and judgmental errors, not necessarily by the pilot. Errors on the part of the designer or air

traffic controller are certainly human. However, pilot error is cited most often [Ref 3: p.13].

Human(pilot) errors can be partitioned into five categories [Ref 4: p.7]:

(1) **Retroactive Interference** - The action of the individual is not identified with the problem at hand. This is motivated by an assimilation of prior input into an ongoing program.

(2) Reductive Coding - An overload of input or a complex stages of events, precludes the correct handling of the situation.

(3) Psychological Refractory Phase - The human unit receives the input. However, a simultaneous transference of this information does not occur. There is a segment of time between the input and output, leaving a window open to information loss or personal interpolation of an event or incoming data.

(4) *Inferential Shortcomings* - Application of *knowledge structures* and *heuristics* to a situation for supposition of data which is non-existent. In short human error due to assumption.

(5) **Leadership and Crew Coordination** - Protocol and the social hierarchy of the aircraft are examined in reference to their role in the cockpit environment.

A single element or a combination of these elements could invoke an error.

The pilot of an aircraft is tasked with monitoring many gauges in the cockpit. A system such as the one proposed could improve pilot effectiveness to some degree in all five categories listed above, particularly in the areas of ***reductive coding*** and ***inferential shortcomings***. The system continuously monitors the gauges and is kept abreast of the aircraft systems they represent. In case of an emergency, any problem that is presented to the pilot is also presented

to the system. Recommended actions are returned to the pilot, by the system, for cross reference or confirmation.

B. OBJECTIVE

The overall objective is to raise to a higher level of abstraction the real-time performance data available to the pilot, utilizing artificial intelligence techniques. The objectives in particular are as follows:

(1) Show that useful information can be provided to the pilot, in terms of procedural recommendations and diagnosis, rapidly with the existing sensor input.

(2) Quantify, through implementation, the amount of information processing necessary to sufficiently define the aircraft emergencies [Ref 5: p.2].

(3) Show the feasibility of a system with multiple knowledge bases.

(4) Show that *means-ends analysis* is an appropriate problem solving formalism on which to solve the problem.

C. SCOPE

This system is implemented around the United States Marine Corps AH-1T[1] helicopter. It demonstrates the ability of this problem solving paradigm to diagnose compound aircraft emergencies and present recommendations to the pilot, in the context of the objectives stated above. Twenty categories of inflight emergencies, for the AH-1T, are

----

[1] The AH-1T is a tandem seat, two place (pilot and copilot/gunner) twin engine attack helicopter manufactured by Bell Helicopter Textron.

listed in the **NATOPS Flight Manual**[1] One category, engine malfunctions, is concentrated on.

D. ASSUMPTIONS

Three assumptions have been made to simplify implementation:

(1) No discrepancies exist between the gauges and caution lights. Caution lights illuminate to show fault conditions. They are housed together in a cluster called a master caution panel. It is assumed that these lights are not faulty.

(2) All indications presented to the pilot are correct. As a result of this, no emergency is caused by a faulty gauge.

(3) The aircraft is operating in the high altitude environment.

E. MATERIALS AND EQUIPMENT

The design was implemented in Turbo Prolog [Ref.6 p.1] on an IBM AT computer.

----

[1] The **NATOPS Flight Manual** is issued by the authority of the Chief of Naval Operations and under the direction of Commander, Naval Air Systems Command in conjunction with the Naval Air Training and Operating Procedures Standardization (NATOPS) Program. This manual standardizes ground and flight procedures based on professional knowledge and experience. Compliance with this manual is mandatory except where stated within the manual.

II. SUMMARY OF CURRENT KNOWLEDGE

A. INDUSTRY

Substantial emphasis in AI developments to date has remained in the military/industrial area.

> Accelerating activity in artificial intelligence and expert systems is pressing researchers to take a long look ahead toward real-world applications in a variety of aerospace systems, where the technology holds promise of enhancing human capabilities. [Ref 2: p.40]

One major driver is the funding and research base provided by the Defense Advanced Research Projects Agency's (DARPA) Strategic Computing Initiative and, more specifically, particular areas targeted by DARPA for initial technology applications [Ref.7 p. 46].

1. Pilot Aids

Development of the Pilot's Associate, an intelligent, personalized airborne system, has moved ahead significantly with award of three-year Phase 1 contracts to two industry teams headed by Lockheed-Georgia Co. and McDonnell Aircraft Co. [Ref.8 p. 34]. Initially, it will consist of four interactive expert systems [Ref.4: p.47]:

(1) A Situation Assessment Manager to assess the external environment as well as internal resources.

(2) A Tactical Planning Manager to recommend optimum tactical employment of the aircraft, given mission objectives and restrictions.

(3) A Mission Planning Manager to refine and redefine

mission objectives, given current situation, command, and intelligence inputs.

(4) A System Status Manager to monitor and diagnose total system health and current/projected status of all on-board systems.

Texas Instruments research included development of an emergency procedures expert system (EPES), focusing on aiding pilots of USAF/General Dynamics F-16 fighters in certain multiple emergency situations [Ref.9 p.89].

2. Maintenance Expert System

McDonnell Douglas Aircraft Corp. has conducted flight tests of an avionics integrated (AIMES) maintenance expert system aboard a Navy/Mcdonnell Douglas F/A-18. During operation, AIMES monitors the aircraft's mission computers for avionics failure indications and records general data from suspect avionic boxes. The expert system then generates queries based upon the general data. Tests are performed by the system to determine the validity of the queries and a conclusion concerning the failure is reached. During interrogation the system can provide the fault data, name the avionics card that has failed, and detail the reasoning that lead to the fault isolation conclusion [Ref.2 p.69].

3. Space Station Operations

The Boeing Aerospace Co. is addressing expert systems in space station operations, and expert systems as pilot decision aids. Expert systems development for the manned space station is focused on the automation of

housekeeping functions, one of which is electrical power control, with emphasis on fault detection and isolation and energy management. The system is optimized for diagnosing multiple or simultaneous faults. In pilot decision aids, Boeing is applying expert systems technology to the pictorial format cockpit display it is developing for the Air Force Systems Command's Aeronautical Systems Div. The display system, called the crew information manager, will use picture symbols instead of numbers to present real-time flight and mission information for all-weather operations in a variety of military aircraft. Boeing expects that work on the system will continue under DARPA's Pilot's Associate program when the current contract expires [Ref.2: p.79].

B. ACADEMIA

Planners are but one of many types of expert systems. The expert system proposed in this paper is of the planner type. There are four basic approaches to planning: ***hierarchical, non-hierarchical, script-based,*** and ***opportunistic.***

1. *Non-hierarchical Planners*

Hierarchical is interpreted as having a hierarchy of representations of a plan in which the highest is a simplification, or *abstraction,* of the plan and the lowest is a detailed plan, sufficient to solve the problem. A non-hierarchical planner develops a sequence of problem-solving

actions to achieve each of its goals and usually has only one representation of a plan. Some examples of non-hierarchical planners are **STRIPS** [Ref.10 p.523] and **HACKER** [Ref.10 p.531]. HACKER generates initial plans that violate ordering constraints and then tries to go back and them. This is based on a *Linearity assumption*, which is that subgoals are independent and thus can be sequentially achieved in an arbitrary order. The Linearity assumption is used in cases where there is no a priori reason to order one operator ahead of another [Ref.10: p. 520]. This assumption could not be made for this implementation. The actions taken by the pilot must be ordered and thus the recommendations to the pilot by the system must be ordered. A characteristic of non-hierarchical planners is the inability to distinguish between the relative importance of recommended actions.

Means-ends analysis is often used in non-hierarchical planners [Ref.10: p. 517], but is also considered by many to be a hierarchical itself. Means-ends is appropriate when it is known how each problem-solving operator changes the state of the world and knows the preconditions for an operator to be executed [Ref.10: p. 524]. Operators or actions are selected according to their ability to reduce the observed *difference* between the *current state* and the *goal state*. [Ref.5: p.147].

> . . . to select these operators, means-ends analysis must be provided with a table listing the best operator for classes of states. These tables refer to the difference

between the current state and the goal state, and are thus called *difference tables.*

Difference tables provide a way of decomposing a hard problem into simpler sub-problems recursively, thus making means-ends a recursive search. Means-ends analysis is also called *hierarchical reasoning*. [Ref.11: p. 11.1]

2. Hierarchical Planners

Hierarchical planners utilize a hierarchy of representations of a plan and are designed to solve some of the problems with nonhierarchical planners. Examples are **NOAH** [Ref.10 p.541], **MOLGEN,** [Ref.10 p.551] and **ABSTRIPS** [Ref.10 p.523]. First a plan is sketched out. The initial sketch, even though complete, is usually vague. Those parts that are vague are refined into more detailed sub-plans until finally the plan has been refined to a complete sequence of detailed problem-solving operators. The major advantage to this is that it provides a means of ignoring the details that obscure or complicate a problem [Ref.10 p. 517]. Because these planners are able to represent a problem at different levels of abstraction, they tend to be very elaborate but effective planning models.

3. Script-Based Planners

The *script-based* method utilizes stored plans which contain the outlines for solving different kinds of problems over a range of classes. One of the **MOLGEN** systems was implemented in this manner. First a skeleton plan is found that is applicable to the given problem. Then the abstract steps in the plan are filled in with problem-solving

operators from the particular problem context. If this can be done for each abstracted step, then the plan as a whole will be successful [Ref.10: p. 518]. Operators are not ordered until constraints are available to guide ordering. This eliminates premature commitment that could cause a conflict with other parts of the plan. Scripts are not well suited to diagnosis of compound emergencies because all scripts can't be anticipated and thus pre-written.

4. <u>Opportunistic Planners</u>

Opportunistic planners are different from those discussed thus far. Operators, or steps in the plan are introduced whenever the opportunity arises. This contrasts greatly with the *least commitment* strategies in NOAH and MOLGEN. Another characteristic of these planners is multidirectionality. Planning takes place on several levels simultaneously.

CRYSALIS is an example of an opportunistic planner. It uses a ***blackboard*** type data-structure to represent the complex control structure of human planning. This involves having a number of ***specialist*** programs that produce hypothesis about data posted on the blackboard. These hypotheses are available to all other specialists.

The blackboard is divided into ***planes***. Planes are organized to reflect characteristic processes in planning. The five categories of planes are (a) the ***plan plane*** which is the actual plan or executive decisions; (b) the ***meta-plan***

plane contains information on the general approach, such as designating means-ends or some other approach; (c) the ***plan abstraction*** plane which contains desirable actions in general, and controls the plan plane; (d) the ***knowledge-base plane*** which contains world or external knowledge; and (e) the ***executive plane*** schedules the planning decisions made by the blackboard [Ref.10: p. 25].

The disadvantage of this method is that it is more likely to rewrite parts of its plan or change its goals than is a hierarchical planner. This takes up valuable time when dealing with real-time constraints.

III. CONCEPT OF THE PLANNER

This planner is passive taking no action on its own. Its recommendations to the pilot would appear on a digital display on the instrument panel. This system could be a replacement for, or supplement to, the master caution panel.

The planner is autonomous in detecting its own goals based on the current ***state description***, rather than simply having the goals handed to it. The system initializes the goals to the empty list, and checks the gauges and caution lights for the initial establishment of goals. Any fluctuations, increases or decreases in ***component status***, require a response.

A means-end control structure does several things:

(1) Means-ends analysis attempts to reduce the difference between the current state and the goal state.

(2) Subgoals are created via problem reduction.

(3) Planning is incorporated by deferring actions until after the overall solution path is established.

A. KERNEL

The means-ends control structure was written, in prolog, by Dr. Neil C. Rowe [Ref.11 p.11.3]. This control structure, and the process of goal acquision, make up the The ***kernel*** of the planner. Once the goals have been acquired and resolved, the facts, in a knowledge-base set along with

means-ends-analysis, are used to satisfy these goals. As with most expert systems, the size of the kernel is small in comparison to the facts in the knowledge-base.

B. KNOWLEDGE-BASE SETS

The planner consists of multiple ***knowledge-base sets*** that enable it to efficiently satisfy goals and to appropriately respond to different requirements in different flight regimes. A knowledge-base set consists of:

(1) *Recommended operators* for achieving goals.

(2) The *preconditions* for the usage of operators.

(3) The effects(*postconditions*) on the state description as a result of the application of operators.

The partitioning of the knowledge-base is critical to both efficiency and correctness. What's recommended for a goal in a high altitude environment could be different from that in a low altitude environment. As a result, different knowledge-base sets are required.

The concept of multiple knowledge-base sets is very similar to an air traffic controllers' handling of aircraft in different control sectors. Once the aircraft leaves a given knowledge-base set's domain, a different knowledge-base set is asserted.

A response to an emergency in a helicopter is in many cases predicated on the aircraft's altitude and airspeed.

Domains defined in terms of altitude or airspeed can be made active when the aircraft enters its domain.

Although this system requires multiple knowledge-base sets, only one is implemented. Only the knowledge-base would need to be changed to adapt it to another aircraft.

IV. IMPLEMENTATION OF THE PLANNER

This chapter describes what techniques were used to construct the planner and why this particular implementation was chosen.

A. PROGRAMMING LANGUAGE

Lisp is the most widely used programming language in artificial intelligence today. However, Prolog, a relatively newer language, is gaining in popularity.

> Prolog has three positive features that give it key advantages over Lisp. First, Prolog in syntax and semantics is much closer to formal logic. The programs are better understood and better maintained. Second, Prolog provides automatic backtracking, a feature that simplifies the writing of *search routines*. Third, Prolog allows a procedure definition to be used for many different kinds of reasoning by allowing the designated input and output parameters to vary from call to call. [Ref.11: preface]

The availability of a Prolog compiler for the IBM AT was a very important factor in this implementation. The run time required to provide the recommended operators for a prescribed goal was of great interest. Turbo Prolog is one of the fastest of the implementations developed for the IBM AT and compatibles [Ref.12: p. 254].

B. APPROACH

First, it is necessary to discuss the *data types* and *symbols* used in the program.

The caution panel cluster is represented by symbols each corresponding to a single caution light or *segment* in the cluster. A list of these segments is a *segment_list*. These segments are shown in Figure 4.1.

```
segment =
            engl_oil_press      eng2_oil_press
            engl_chip_detr      eng2_chip_detr
            engl_fuel_filter    eng2_fuel_filter
            dc_gen_1            dc_gen_2
            xmsn_chip_detr      c_box_chip_detr
            temp_press_90       temp_press_42
            chip_detr_90        chip_detr_42
            xmsn_oil_hot        xmsn_oil_press
            c_box_oil_press     c_box_oil_hot
            hyd_press_1         hyd_press_2
            hyd_temp_1          hyd_temp_2
            ac_main             ac_stby
            engl_gov_man        eng2_gov_man
            fire_1_pull         fire_2_pull
            rpm_rotor_low       rpm_rotor_high
            rpm_ngl             rpm_ng2
            xmsn_oil_byp
```

Figure 4.1 Segment Symbols Representing Caution Panel

A partial state description of the aircraft and its environment at any given time is defined in terms of ***state_elements***. A list of state elements is a ***state_list***. State elements are shown in Figure 4.2

1. <u>Goal Acquisition</u>

The problem solver has access to the instruments and caution lights and evaluates them in terms of their

status, i.e. high, low, erratic etc. Any changes in status are flagged as requiring a *response*. The aircraft's instruments are referred to as *components*. However, a component is defined as being any of the symbols listed in Figure 4.3.

```
state_element =

  full(component)                 landing_zone(zone_quality)
  metal_particles(component) automatic(component)
  manual(component)               idle(component)
  normal(component)               decrease(component)
  failed(component)               hot(component)
  on(component)                   engaged(component)
  off(component)                  fire(component)
  gross_weight(wt_class)          airspeed(knots)
  altitude(ft)                    oat(component_status)
  land(type_landing)              oil_bypassing_cooler
  left_yaw             fuel_obstruction(component)
  power(component_status)  fuel_press(component_status)
  ammeter(component,component_status)
  revs( component, component_status)
  oil_press( component, component_status)
  oil_temp( component, component_status )
  torque( component, component_status )
  prepare_for_failure(component)
  itt(component,component_status)
```

Figure 4.2 State Elements

The goals are generated by scanning each caution light segment in the master caution panel and each instrument, as seen by the pilot. For each caution light there is a set of goals. Prior to assertion, the goals are checked against all the other goals asserted previously.

Duplicates, and goals which are subordinate to previous goals (in its class), are not retained. Classes of goals are defined by ***priority sets*** which list explicitly the relative priority of each goal in a given class. If a goal of a higher priority has been asserted, no lower priority goals will be asserted. Figure 4.4 shows an example of the relative priority between ***power*** status values. In most cases a specific goal belongs to only one class or (set). The majority of the priority sets contain only one goal.

---

component =

eng1	*eng2*	*xmsn*
c_box	*eng_oil_press*	*gen1*
gen2	*gov1*	*gov2*
master_caution	*master_arm*	*ecu*
scas	*fuel*	*fuel1*
fuel2	*fuel_press*	*throttle1*
throttle2	*oil_temp_90*	*oil_press_90*
oil_temp_42	*oil_press_42*	*gear_box_90*
gear_box_42	*c_box_oil_temp*	*c_box_oil_press*
xmsn_oil_temp	*xmsn_oil_press*	*hyd_sys_1*
hyd_sys_2	*hyd_press*	*hyd_temp_1*
hyd_temp_2	*rotor*	*ng1*
ng2	*itt_1*	*itt_2*
nf1	*nf2*	*rain_rmv*
power	*main_inverter*	*standby_inverter*

Figure 4.3 Aircraft Component Symbols

A component is described in terms of its ***component_status***. One or more of eight possible status

values may apply to a given component. The goals generated by the planner must be free of conflicts. A possible conflict exists, for a given component, when two or more status values are true at the same time. These status values are prioritized by facts in the knowledge base via a prioritized *status_list*, which is a list of *component status* symbols. Component status values are shown in Figure 4.4.

```
        component_status =

                none      high      low
                increase  decrease  ok
                erratic   respond

    priority_set([power(none),      power(low),
                  power(decrease),  power(increase)]).
```

Figure 4.4 Component Status Values

At the highest level of abstraction, the goals passed to means-ends are merely to *respond* to a given gauge. The *state_list* is then searched for the specific status of the component. The *respond* status is slightly different from the other quantifying and mutually exclusive status labels such as *low, high* etc. For a given component, the status *respond* can co-exist with a high or low component status in the current state description.

2. Organizing the Search

The order in which goals are selected from a goal list to be satisfied is driven by the order of the ***recommended*** facts. The most important goals are placed first. If there is more than one ***recommended*** operator that applies to a given goal then those operators are ordered according to the urgency of the operator. An example of such *recommended* facts are shown in Figure 4.5.

Recommended facts with multiple goals should have the goal that is least likely to be satisfied first in its goal list. Search time can be reduced by explicitly including predicted response groups into the *recommended* structure. An example of this is the *confirm_dual_eng_failure* operator.

```
recommended( State, [revs(ng1,respond),
            revs(ng2,respond), off(gen1),
            off(gen2), off(eng1), off(eng2),
            land(pract)], confirm_dual_eng_failure).

recommended( State, [torque(eng1, respond)],
             confirm_nf1_failure).

recommended( State, [torque(eng1, respond)]
            check_ng1_overspeed).

recommended( State, [torque(eng1, respond)]
            check_ng1_underspeed).
```

Figure 4.5 Recommended Facts

Grouping multiple goals in one recommendation reduces the number of times a given operator is invoked unnecessarily. However, this should not be a substitute for the single goal *recommended* facts. Single goal *recommended* facts are the finest granularity of the search process.

The output of this planner is to present the *operators* to the pilot as recommendations to satisfy an immediate goal. A list of these operators are referred to as an *op_list*. Operators are shown in Figure 4.6.

---

operator =

normal_approach	*master_arm_off*
slope_landing	*confirm_dual_eng_failure*
steep_approach	*confirm_eng1_failure*
high_speed_approach	*confirm_eng2_failure*
max_weight_landing	*engage_scas*
sliding_landing	*ecu_off*
pract_landing	*rain_rmv_off*
pract_landing_outranked1	*gov1_to_manual*
pract_landing_outranked2	*gov2_to_manual*
pos_landing1	*gov1_to_automatic*
pos_landing2	*gov2_to_automatic*
pos_landing_outranked	*confirm_nf1_failure*
autorotation	*confirm_nf2_failure*
check_ng1_overspeed	*throttle1_idle*
check_ng1_underspeed	*throttle2_idle*
check_ng2_overspeed	*check_gen1_failure*
check_ng2_underspeed	*check_gen2_failure*
gen1_off	*eng1_off*
gen2_off	*eng2_off*

Figure 4.6 Recommended Operators

There are nine different landing profiles and emergencies. A landing as a result of a malfunction could be described as land as soon as practical, soon as possible, or a landing via autorotation. With the exception of an autorotation, landings are a function of aircraft *wt_class* or landing *zone_quality*. Landing profiles and classes are shown in Figure 4.7.

```
wt_class =
        heavy ; moderate

zone_quality =
            clear ; confined ; slope

type_landing =
            pract; pos; auto;
            normal; high_speed; slope;
            sliding; max_weight; steep

priority_set([land(pos), land(pract)]).
```

Figure 4.7 Landing Profiles

3. <u>Natops to Code Translation</u>

The NATOPS manual together with the experience of the author played the role of the *expert* for this system. It was necessary to translate the procedures and system information contained in the NATOPS manual into a knowledge-base. Examples of this translation are now discussed. In

each example the indications and corrective action are taken from the NATOPS manual.

The first example, shown in Figure 4.8 describes the initial action required by the pilot in the case of a *chip*

Caution and warning light - INITIAL ACTION

Panel wording: CHIP DETR

Condition: metal particles in engine

Corrective Action: Flight idle. Check oil pressure and temperature. If normal operate at reduce power. If pressure is low and/or temperature is high, shut down respective engine. Land as soon as practical.

This information is coded as follows...

```
caution_light( State,  engl_chip_detr ):-
    member(metal_particles(engl), State ),
    write( " Caution light: engl_chip_detr " ),nl,
    secondary(State, engl_chip_detr).

secondary(State, engl_chip_detr) :-
    member(oil_press(engl, X ), State ),
    below_limit(L), status_member(X,L),
    create_goals([ off(engl),land(pract)] ).

secondary(State, engl_chip_detr) :-
    member(oil_temp(engl, high), State ),
    create_goals([ off(engl),land(pract)] ).

secondary(State, engl_chip_detr) :-
    create_goals([ idle(throttle1),land(pract)] ).
```

Figure 4.8 Translation Example I

detector caution light. By assumption, caution lights and instruments do not need to be cross checked. In this example there is more than one course of action. Depending on the status of the oil pressure and temperature, (secondary indications), different goals are asserted. The goals that are created are thus a function of the primary(caution light) and secondary indications. The predicate *create_goals* conditionally asserts the goals that are its arguments based on goal priority and duplication.

Figure 4.9 shows the translation of a definition of a power turbine governor failure into code. If the immediate goal is to respond to the torque of the #1 engine then it would be recommended to check the #1 nf governor. If a precondition to some operator is that the #1 governor be in a failed state, and this has yet to be proven, then the second recommended fact would be used.

The indications of a #1 nf governor failure are represented by the preconditions listed in Figure 4.9. If the preconditions are true then nothing will be deleted from the current state description because nothing has changed. Only a diagnosis of the situation has been accomplished. Nothing has been done about it at this point. The current state description is amended to reflect the fact that the governor has failed and the instruments listed in the *addpostcondition* fact have been responded to.

Power Turbine Governor (NF) Failure

Indications.

1. Erratic GAS PROD RPM (Ng).
2. Erratic INLET TEMP.
3. Fluctuating ENG RPM (Nf).
4. abrupt increase in ENG RPM (Nf) above governed value.
5. Abrupt decrease in ENG RPM (Nf) below governed value.
6. Fluctuating TORQUE.

Procedure.

1. Affected engine - IDENTIFY
2. Throttle - ENGINE IDLE.
3. GOV - MANUAL
4. Throttle - ADVANCE.
5. LAND AS SOON AS PRACTICAL.

This information is coded as follows...

```
recommended( State, [torque(eng1,respond)],
    confirm_nf1_failure).

recommended( State, [failed(gov1)],
    confirm_nf1_failure).

precondition( State, confirm_nf1_failure,
    [revs(ng1,erratic), itt(eng1,erratic),
     revs(nf1,erratic), torque(eng1,erratic),
     automatic(gov1)])

deletepostcondition( State,
     confirm_nf1_failure,[]).

addpostcondition( confirm_nf1_failure,
     [revs(ng1,respond), revs(nf1,respond),
     failed(gov1),torque(eng1,respond)]).
```

Figure 4.9 Translation Example II

The third example, a ***single engine failure,*** is shown in Figure 4.10. When an engine failure occurs the rotor speed decreases somewhat due to loss of power. This creates the goal ***revs(rotor, respond)***. One of the recommended operators for this goal is ***confirm_eng1_failure***. If all of the preconditions have been meet, the current state description will be altered by the addition and deletion of facts as shown.

Single Engine Failure (In Flight).

When one engine fails, rotor speed can be expected to droop. The desired rotor rpm can be regained if sufficient power is available, by using the engine RPM switch. After rpm is regained by use of the RPM switch, desired rotor rpm can be maintained by the collective control.

INDICATIONS.
1. Left yaw
2. RPM caution light (gas producer)
3. MASTER CAUTION light
4. Rotor rpm decrease
5. Engine instruments decrease
   { engine_instruments }
6. CAUTION panel lights

This information is coded as follows...

```
recommended( State,
    [revs(rotor,respond)],  confirm_eng1_failure).

precondition( State, confirm_eng1_failure,
    [torque(eng1,none), torque(xmsn,low),
    revs(ng1,none), revs(nf1,none),
    oil_press(eng1,none), failed(gen1),
    itt(eng1,none), left_yaw, on(fuel1),
    full(throttle1), revs(rotor,low)]).

deletepostcondition( State,  confirm_eng1_failure,
    [on(ecu)] ).

addpostcondition( confirm_eng1_failure,
    [ off(ecu),failed(eng1),revs(ng1,respond),
     revs(nf1,respond),revs(rotor,respond),
     itt(eng1,respond),torque(eng1,respond),
     torque(xmsn,respond)]).
```

Figure 4.10 Translation Example III

V. ANALYSIS AND CONCLUSIONS

A. ANALYSIS

A version of the planner was tested to determine the execution time of the various test cases. Only the intermediate write statements, included in the source code in appendix D, were left out. These times are interesting because they give some relative measurement to the costs of the various aspects of the planner. The response times for each test are shown in Figure 5.1.

Detailed output listings of tests 2, 3, and 4, can be found in appendices A, B, and C respectfully.

TEST #	TIME	EMERGENCY
1	0.39 sec.	*none*
2	1.82 sec.	*dual engine failure*
3	1.92 sec.	*engine #1 failure*
4	2.53 sec.	*engine #1 failure* and *#2 Nf governor failure*

Figure 5.1 Test Case Response Times

1. Test 1

This test involved no goals. It did provide the amount of time necessary to scan the current state

description. The base time of 0.39 sec. as reflected by test #1 is also significant because this determines the sampling rate of the planner. The state description then could be sampled approx. 150 times per minute. The longest reaction time is experienced when a component failure occurs immediately **after** a sampling of the current state description. This means as much as 0.8 sec. could elapse before a response condition would be acted upon by the planner.

2. <u>Test 2</u>

In test #2 only four operators are considered and all four were applicable. Figure 5.2 shows, through indentation, the nesting levels for each operator invoked. Backtracking occurs when the indentation is reversed, and no backtracking occurs in this example. This is the optimum situation. Here the planner is telling the pilot that it has confirmed a dual engine failure and an autorotation is to be accomplished.

3. <u>Test 3</u>

In this case, 11 operators were tried with 5 being applicable. Figure 5.2 depicts the backtracking that occurs as a result of some recommended operators failing to satisfy immediate goals. It should be pointed out that two of the operators, *confirm_eng2_failure* and *check_gen2_failure*, were attempted twice. The pilot in this case is informed that the

---

TEST #1

-no recommended operators-

TEST #2

```
confirm_dual_eng_failure
 pract_landing_outranked1
  pos_landing_outranked
   autorotation
```

TEST #3

```
confirm_eng2_failure
 check_gen2_failure
confirm_eng1_failure
 eng1_off
  gen1_off
   pract_landing_outranked1
    pos_landing_outranked
     autorotation
      confirm_eng2_failure
       check_gen2_failure
    pos_landing
```

TEST #4

```
confirm_eng2_failure
 check_gen2_failure
confirm_eng1_failure
 confirm_nf2_failure
  eng1_off
   gen1_off
    pract_landing_outranked1
     pos_landing_outranked
      autorotation
       confirm_eng2_failure
        check_gen2_failure
     pos_landing
```

Figure 5.2 Recommended Operator Selection

#1 engine has failed, to secure the #1 engine and generator and to land as soon as possible.

4. <u>Test 4</u>

Here 12 operators were selected with 6 being applicable. Again, two of the operators, *confirm_eng2_failure* and *check_gen2_failure*, were attempted twice. In addition to the information presented to the pilot in test 3, the pilot is informed that the #2 nf-governor has failed.

Governor failures are not as straightforward to diagnose as engine failures. As a result this information is potentially more valuable.

5. <u>Programming Language</u>

Turbo Prolog is not considered to be a Clocksin and Mellish PROLOG as most others are [Ref.13 p.334]. The first major difference between Turbo Prolog and other implementations is the required use of Pascal-like type definitions for parameters. This has the advantage of catching various errors at compile time, and also allows the compiler to generate more efficient code. The drawback is that describing generalized procedures can sometimes result in multiple definitions of a rule to handle different types of variables. An example of this was the *member* predicate. As defined, it could accept as arguments a *state_element* and a *state_list*. However, when this predicate was needed to be used with arguments of type *component_status* and *status_list*

respectfully, a separate definition of the member predicate had to be defined named "status_member". The Pascal-like syntax was more of a help than a hindrance and greatly helped in the debugging process.

B. CONCLUSIONS

Turbo Prolog proved to be suitable for this purpose and the timing results were encouraging. Useful information was rapidly presented in the form of diagnosis and recommended actions.

Means-ends proved to be effective because the structure of the task was well suited to the data structure of the *recommended* facts. The amount of information processing required to satisfy the various intermediate goals is reflected by the number of recommended operators considered in satisfying the immediate goals. All of the facts in the knowledge-base were not utilized by the examples exercised in this implementation.

Translation from the NATOPS manual into a knowledge-base was straightforward and easily verified. In employing multiple knowledge-base sets, correctness and speed of execution can be maintained.

It is conceivable that this system could be used in place of the existing master caution panel which displays caution light information only. Perhaps a digital display that presents graphically, as well as textually, the

operators and actions to be taken. In this way no extra space would need to be provided for this system in the cockpit. This makes this feasible for use in existing aircraft.

For total aircraft system implementation, the number of operators required is dependent on the granularity of information provided to the pilot. Higher levels of abstraction are required for immediate actions, with more detailed instructions provided at the request of the pilot. Different help levels could be provided by switching to a specialized knowledge-base. This implies an interactive capability by the pilot to select goals, and a means to do so would need to be provided.

The inflight emergency domain requires as a minimum the following domains: Takeoff, Low altitude, High altitude, Night, and perhaps instrument conditions.

This system potentially lends itself to practically any application within the aircraft. The task of piloting an aircraft is very well defined and procedurized and could possibly be expanded into the realm of normal procedures as well as emergency procedures.

C. RECOMMENDATIONS

It is recommended that a full implementation be built to determine the number of knowledge-base sets needed and the scope of each.

APPENDIX A

DUAL ENGINE FAILURE EXECUTION

```
 Caution light: eng1_oil_press
 Caution light: eng2_oil_press
 Caution light: dc_gen_1
 Caution light: dc_gen_2
 Caution light: rpm_rotor_low
 Caution light: rpm_ng1
 Caution light: rpm_ng2
```

Goal list is: [revs(ng2,respond), revs(ng1,respond), revs(rotor,respond), off(gen2), off(gen1), off(eng2), land(pract), off(eng1)]

difference: [revs(ng2,respond), revs(ng1,respond), revs(rotor,respond), off(gen2), off(gen1), off(eng2), land(pract), off(eng1)]

```
***********************************************
current operator: confirm_dual_eng_failure    depth is: 1
***********************************************
```

operator preconditions have been met

items being deleted are: [on(ecu)]

items being added are: [off(ecu), failed(eng1), failed(eng2), off(gen1), off(gen2), off(eng1), off(eng2), revs(ng1,respond), revs(nf1,respond), revs(rotor,respond), itt(eng1,respond), itt(eng2,respond), revs(ng2,respond), revs(nf2,respond)]

Goal list is: [revs(ng2,respond), revs(ng1,respond), revs(rotor,respond), off(gen2), off(gen1), off(eng2), land(pract), off(eng1)]

difference: [land(pract)]

```
***********************************************
current operator: pract_landing_outranked1    depth is: 2
***********************************************
```

Goal list is: [land(pos)]

difference: [land(pos)]

****

current operator: pos_landing_outranked   depth is: 3

****

Goal list is: [land(auto)]

difference: [land(auto)]

****

current operator: autorotation   depth is: 4

****

operator preconditions have been met

items being deleted are: [full(throttle1), full(throttle2), airspeed(100), altitude(1000), revs(rotor,respond)]

items being added are: [idle(throttle1), idle(throttle2), airspeed(0), altitude(0), revs(rotor,respond),land(auto)]

operator preconditions have been met

items being deleted are: []

items being added are: [land(pos)]

operator preconditions have been met

items being deleted are: []

items being added are: [land(pract)]

operator preconditions have been met

The recommended operators are: [confirm_dual_eng_failure, autorotation, pos_landing_outranked, pract_landing_outranked1]

The final state description is: [oil_press(eng1,none), oil_press(eng2,none), revs(rotor,low), off(rain_rmv), off(master_arm), landing_zone(clear), full(throttle1),

```
full(throttle2),    off(master_caution),    automatic(gov1),
automatic(gov2),  on(fuel1),  on(fuel2), normal(fuel_press),
torque(eng1,none),    torque(eng2,none),   torque(xmsn,low),
revs(ng1,none),         revs(ng2,none),        revs(nf1,none),
revs(nf2,none),     oil_temp(eng1,ok),    oil_temp(eng2,ok),
failed(gen1),  failed(gen2), full(fuel), off(scas), on(ecu),
oil_temp(c_box,ok), oil_press(c_box,ok), oil _temp(xmsn,ok),
oil_press(xmsn,ok), ammeter(gen1,none), ammeter (gen2,none),
fuel_press(ok),          itt(eng1,none),        itt(eng2,none),
oil_press(hyd_sys_1,ok),               oil_press(hyd_sys_2,ok),
airspeed(100), altitude(1000), left_yaw]
```

APPENDIX B

SINGLE ENGINE FAILURE EXECUTION

```
 Caution light: eng1_oil_press
 Caution light: dc_gen_1
 Caution light: rpm_rotor_low
 Caution light: rpm_ng1

Goal list is: [revs(ng1,respond), revs(rotor,respond),
off(gen1), land(pract), off(eng1)]

difference: [revs(ng1,respond), revs(rotor,respond),
off(gen1), land(pract), off(eng1)]

***********************************************

current operator: confirm_eng2_failure    depth is: 1

***********************************************

Goal list is: [torque(eng2,none), torque(xmsn,low),
revs(ng2,none), revs(nf2,none), oil_press(eng2,none),
failed(gen2), itt(eng2,none), left_yaw, on(fuel2),
full(throttle2), revs(rotor,low)]

difference: [torque(eng2,none), revs(ng2,none),
revs(nf2,none), oil_press(eng2,none), failed(gen2),
itt(eng2,none)]

***********************************************

current operator: check_gen2_failure    depth is: 2

***********************************************

Goal list is: [on(gen2),ammeter(eng2,none)]

difference: [on(gen2),ammeter(eng2,none)]

***********************************************

current operator: confirm_eng1_failure    depth is: 1

***********************************************

 operator preconditions have been met
```

items being deleted are: [on(ecu)]

items being added are: [off(ecu), failed(engl), revs(ngl,respond), revs(nfl,respond), revs(rotor,respond), itt(engl,respond), torque(engl,respond), torque(xmsn,respond)]

Goal list is: [revs(ngl,respond), revs(rotor,respond), off(genl), land(pract), off(engl)]

difference: [off(genl),land(pract),off(engl)]

****

current operator: engl_off   depth is: 2

****

operator preconditions have been met

items being deleted are: [on(engl)]

items being added are: [off(engl)]

Goal list is: [revs(ngl,respond), revs(rotor,respond), off(genl), land(pract), off(engl)]

difference: [off(genl),land(pract)]

****

current operator: genl_off   depth is: 3

****

operator preconditions have been met

items being deleted are: [on(genl)]

items being added are: [off(genl)]

Goal list is: [revs(ngl,respond), revs(rotor,respond), off(genl), land(pract), off(engl)]

difference: [land(pract)]

****

current operator: pract_landing_outranked1   depth is: 4

```
***********************************************

Goal list is: [land(pos)]

difference: [land(pos)]

***********************************************

current operator: pos_landing_outranked    depth is: 5

***********************************************

Goal list is: [land(auto)]

difference: [land(auto)]

***********************************************

current operator: autorotation    depth is: 6

***********************************************

Goal list is: [failed(eng1),failed(eng2)]

difference: [failed(eng2)]

***********************************************

current operator: confirm_eng2_failure    depth is: 7

***********************************************

Goal   list   is:    [torque(eng2,none),   torque(xmsn,low),
revs(ng2,none),     revs(nf2,none),     oil_press(eng2,none),
failed(gen2),     itt(eng2,none),     left_yaw,     on(fuel2),
full(throttle2), revs(rotor,low)]

difference:        [torque(eng2,none),       revs(ng2,none),
revs(nf2,none),      oil_press(eng2,none),      failed(gen2),
itt(eng2,none)]

***********************************************

current operator: check_gen2_failure    depth is: 8

***********************************************

Goal list is: [on(gen2),ammeter(eng2,none)]

difference: [on(gen2),ammcter(eng2,none)]
```

```
************************************************

current operator: pos_landing1    depth is: 5

************************************************

 operator preconditions have been met

items being deleted are: [altitude(1000),airspeed(100)]

items being added are: [land(pos),altitude(0),airspeed(0)]

 operator preconditions have been met

items being deleted are: []

items being added are: [land(pract)]

 operator preconditions have been met


The   recommended   operators   are:   [confirm_eng1_failure,
eng1_off, gen1_off, pos_landing1, pract_landing_outranked1]


The  final  state  description  is:  [oil_press(eng1,none),
oil_press(eng2,ok),       revs(rotor,low),      off(rain_rmv),
off(master_arm),    landing_zone(clear),    full(throttle1),
full(throttle2),    off(master_caution),    automatic(gov1),
automatic(gov2),  on(fuel1),  on(fuel2), normal(fuel_press),
torque(eng1,none),      torque(eng2,ok),     torque(xmsn,low),
revs(ng1,none),  revs(ng2,ok), revs(nf1,none), revs(nf2,ok),
oil_temp(eng1,ok),      oil_temp(eng2,ok),       failed(gen1),
full(fuel),    off(scas),    on(ecu),    oil_temp(c_box,ok),
oil_press(c_box,ok),  oil_temp(xmsn,ok), oil_press(xmsn,ok),
ammeter(gen1,none),      ammeter(gen2,ok),     fuel_press(ok),
itt(eng1,none),    itt(eng2,ok),    oil_press(hyd_sys_1,ok),
oil_press(hyd_sys_2,ok),    airspeed(100),   altitude(1000),
left_yaw]
```

APPENDIX C

SINGLE ENGINE FAILURE / NF GOVERNOR FAILURE EXECUTION

revs(ng2, erratic)

revs(nf2, erratic)

torque(eng2, erratic)

torque(xmsn, erratic)

Caution light: eng1_oil_press
Caution light: dc_gen_1
Caution light: rpm_rotor_low
Caution light: rpm_ng1

Goal list is: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

difference: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

****

current operator: confirm_eng2_failure   depth is: 1

****

Goal list is: [torque(eng2,none), torque(xmsn,low), revs(ng2,none), revs(nf2,none), oil_press(eng2,none), failed(gen2), itt(eng2,none), left_yaw, on(fuel2), full(throttle2), revs(rotor,low)]

difference: [torque(eng2,none), revs(ng2,none), revs(nf2,none), oil_press(eng2,none), failed(gen2), itt(eng2,none)]

****

current operator: check_gen2_failure   depth is: 2

****

Goal list is: [on(gen2),ammeter(eng2,none)]

difference: [on(gen2),ammeter(eng2,none)]

****

current operator: confirm_eng1_failure   depth is: 1

****

operator preconditions have been met

items being deleted are: [on(ecu)]

items being added are: [off(ecu), failed(eng1), revs(ng1,respond), revs(nf1,respond), revs(rotor,respond), itt(eng1,respond), torque(eng1,respond), torque(xmsn,respond)]

Goal list is: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

difference: [off(gen1), land(pract), off(eng1), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

****

current operator: confirm_nf2_failure   depth is: 2

****

operator preconditions have been met

items being deleted are: []

items being added are: [revs(ng2,respond), revs(nf2,respond), failed(gov2), torque(eng2,respond)]

Goal list is: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

difference: [off(gen1),land(pract),off(eng1)]

****

current operator: eng1_off   depth is: 3

****

operator preconditions have been met

items being deleted are: [on(eng1)]

items being added are: [off(eng1)]

Goal list is: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

difference: [off(gen1),land(pract)]

****

current operator: gen1_off   depth is: 4

****

operator preconditions have been met

items being deleted are: [on(gen1)]

items being added are: [off(gen1)]

Goal list is: [revs(ng1,respond), revs(rotor,respond), off(gen1), land(pract), off(eng1), torque(xmsn,respond), torque(eng2,respond), revs(nf2,respond), revs(ng2,respond)]

difference: [land(pract)]

****

current operator: pract_landing_outranked1   depth is: 5

****

Goal list is: [land(pos)]

difference: [land(pos)]

****

current operator: pos_landing_outranked   depth is: 6

****

Goal list is: [land(auto)]

difference: [land(auto)]

****

```
current operator: autorotation    depth is: 7

**********************************************

Goal list is: [failed(eng1),failed(eng2)]

difference: [failed(eng2)]

**********************************************

current operator: confirm_eng2_failure    depth is: 8

**********************************************

Goal list is: [torque(eng2,none), torque(xmsn,low),
revs(ng2,none), revs(nf2,none), oil_press(eng2,none),
failed(gen2), itt(eng2,none), left_yaw, on(fuel2),
full(throttle2), revs(rotor,low)]

difference: [torque(eng2,none), revs(ng2,none),
revs(nf2,none), oil_press(eng2,none), failed(gen2),
itt(eng2,none)]

**********************************************

current operator: check_gen2_failure    depth is: 9

**********************************************

Goal list is: [on(gen2),ammeter(eng2,none)]

difference: [on(gen2),ammeter(eng2,none)]

**********************************************

current operator: pos_landing1    depth is: 6

**********************************************

 operator preconditions have been met

items being deleted are: [altitude(1000),airspeed(100)]

items being added are: [land(pos),altitude(0),airspeed(0)]

 operator preconditions have been met

items being deleted are: []

items being added are: [land(pract)]
```

operator preconditions have been met

The recommended operators are: [confirm_eng1_failure, confirm_nf2_failure, eng1_off, gen1_off, pos_landing1, pract_landing_outranked1]

The final state description is: [oil_press(eng1,none), oil_press(eng2,ok), revs(rotor,low), off(rain_rmv), off(master_arm), landing_zone(clear), full(throttle1), full(throttle2), off(master_caution), automatic(gov1), automatic(gov2), on(fuel1), on(fuel2), normal(fuel_press), torque(eng1,none), torque(eng2,erratic), torque(xmsn,low), torque(xmsn,erratic), revs(ng1,none), revs(ng2,erratic), revs(nf1,none), revs(nf2,erratic), oil_temp(eng1,ok), oil_temp(eng2,ok), failed(gen1), full(fuel), off(scas), on(ecu), oil_temp(c_box,ok), oil_press(c_box,ok), oil_temp(xmsn,ok), oil_press(xmsn,ok), ammeter(gen1,none), ammeter(gen2,ok), fuel_press(ok), itt(eng1,none), itt(eng2,erratic), oil_press(hyd_sys_1,ok), oil_press(hyd_sys_2,ok), airspeed(100), altitude(1000), left_yaw]

APPENDIX D

SOURCE CODE LISTING

```
nowarnings
code = 3500
trail = 100
/************************ DOMAINS **************************/
domains

/*  The  domains  section contains the declarations  of  the
types and symbols used in this program
*/
/*  The caution panel cluster is represented by  terms  each
corresponding to a single light in the aircraft
*/

  segment =
    eng1_oil_press;     eng2_oil_press;
    eng1_chip_detr;     eng2_chip_detr;
    eng1_fuel_filter;   eng2_fuel_filter;
    dc_gen_1;           dc_gen_2;
    xmsn_chip_detr;     c_box_chip_detr;
    temp_press_90;      temp_press_42;
    chip_detr_90;       chip_detr_42;
    xmsn_oil_hot;       xmsn_oil_press;     xmsn_oil_byp;
    c_box_oil_press;    c_box_oil_hot ;
    hyd_press_1;        hyd_press_2;
    hyd_temp_1;         hyd_temp_2;
    ac_main;            ac_stby ;
    eng1_gov_man;       eng2_gov_man;
    fire_1_pull;        fire_2_pull;
    rpm_rotor_low;      rpm_rotor_high;
    rpm_ng1;            rpm_ng2

  segment_list = segment*

/*
   Aircraft weight and landing zone quality are  represented
by the follow-   ing declarations. with respect to the scope
of  this implementation   the weight can be one of two types
and the zone quality can be one of   three values.  The nine
different landing profiles and ergencies.  A landing   as a
result  of a malfunction could be described as land as  soon
as     practical,   soon  as  possible,  or  a  landing  via
autorotation.  Profiles    can can be  normal,  high  speed,
sliding,  steep, aircraft at max weight,   or a landing to a
```

```
sloped zone.
*/
  wt_class =
          heavy ; moderate

  zone_quality =
              clear ; confined ; slope

  type_landing =
              pract; pos; auto;
              normal; high_speed; slope;
              sliding; max_weight; steep

/*
  A  component  is defined as being any of the below  listed
names.  Some  of these components were not utilized  in  the
knowledge base.
*/
  component =
           eng1 ; eng2 ; xmsn ; c_box ;
           eng_oil_press;
           gen1 ; gen2 ; gov1 ; gov2 ;
           master_caution ; master_arm ;
           ecu ; scas ;
           fuel ; fuel1 ; fuel2 ; fuel_press ;
           throttle1 ; throttle2 ;
           oil_temp_90 ; oil_press_90 ;
           oil_temp_42 ; oil_press_42 ;
           gear_box_90 ; gear_box_42 ;
           c_box_oil_press; c_box_oil_temp;
           xmsn_oil_temp; xmsn_oil_press ;
           hyd_sys_1 ; hyd_sys_2 ; hyd_press;
           hyd_temp_1; hyd_temp_2;
           rotor ; ng1 ; ng2 ; itt_1 ; itt_2 ;
           nf1 ; nf2 ; rain_rmv ; power ;
           main_inverter ; standby_inverter

/*  The  aircraft's  guages and instruments as seen  by  the
pilot are refered to  as components.  A component has eight
status  values  and  are  prioritized   by  a  fact  in  the
knowledge base.  A given component may have more than    one
status true at any given time. For instance eng torque could
be    both 'ok' and 'decrease' simultaniously.
*/
  component_status =
           none ; high ; low ;
           increase ; decrease ;
           ok ; erratic ; respond

/*
   A status list is a list of component status values.  This
```

```
list    established the priority of the status values
*/
  status_list = component_status*

/*  Some   components   are described as being in one   of   the
following two   modes. Each is mutually exclusive
*/
  manual,
  automatic = mode

/*  A  partial state description of the  aircraft  and  it's
environment  at    any given time is defined in terms of the
following state elements.
*/
 state_element =
 prepare_for_failure(component);power(component_status);
 full(component);                landing_zone(zone_quality);
 metal_particles(component);     fuel_obstruction(component);
 fuel_press(component_status);   automatic(component);
 manual(component);              idle(component);
 normal(component);              decrease(component);
 failed(component);              hot(component);
 on(component);                  engaged(component);
 off(component);                 fire(component);
 gross_weight(wt_class);         airspeed(knots);
 altitude(ft);                   oat(component_status);
 land(type_landing);             oil_bypassing_cooler;
 itt(component,component_status); left_yaw;
 ammeter(component,component_status);
 revs( component, component_status);
 oil_press( component, component_status);
 oil_temp( component, component_status );
 torque( component, component_status )

 state_list = state_element*

/*  Actions  to be taken by the pilot or some  action  which
must be taken   to satisfy an immediate goal.  The objective
of  this planner is to   present the operators to the  pilot
as a possible diagnosis of the   present state of affairs. A
list of these operators are refered to as   an op_list.
*/
  operator =
 normal_approach ; slope_landing ; steep_approach ;
 high_speed_approach ; max_weight_landing ; sliding_landing;
 pract_landing;pract_landing_outranked1;
 pract_landing_outranked2;pos_landing1;pos_landing2;
 autorotation;confirm_eng1_failure; confirm_eng2_failure;
 engage_scas ; ecu_off ; rain_rmv_off ; master_arm_off;
 gov1_to_manual;gov1_to_automatic;confirm_nf1_failure;
 check_ng1_overspeed ; check_ng1_underspeed ;gov2_to_manual;
```

```
 gov2_to_automatic ;confirm_nf2_failure;check_ng2_overspeed;
 check_ng2_underspeed ; throttle1_idle ; throttle2_idle;
 confirm_dual_eng_failure ; check_gen1_failure ;
 check_gen2_failure;gen1_off; gen2_off;  eng1_off; eng2_off;
 pos_landing_outranked

  op_list = operator*

/*********************** DATABASE ************************/
database

/*  The  database is contains only one fact and that is  the
list  of goals    that are asserted by scanning the  caution
lights  and  the instrument  panel. The database  is  only
active  during  this process and the    fact  temp_goals  is
retracted prior to calling means_ends_analysis.
*/
  temp_goals          (state_list)


  nest_level,
  percent, ft, degrees, amps,
  lbs, knots, oat, value
  = integer
```

```
/*********************** PREDICATES ***********************/
predicates

  problem_solver      ( state_list,op_list,state_list )

/* predicates used directly in the means ends analysis
procedure
*/
    means_ends          ( state_list,state_list,op_list,
                          state_list,nest_level )
    recommended         ( state_list, state_list, operator )
    precondition        ( state_list, operator, state_list )
    addpostcondition    ( operator, state_list )
    deletepostcondition ( state_list, operator, state_list )

/* predicates used to scan the cockpit for goals
*/
  check_lights          ( state_list, state_list )
  check_guages          ( state_list )
    check_revs          ( state_list )
    check_pressures     ( state_list )
    check_temps         ( state_list )
    check_torque        ( state_list )

  master_caution        ( state_list, state_list )
  caution_light         ( state_list, segment )
  secondary             ( state_list, segment)

  create_goals          ( state_list )
    ck_duplicate        ( state_element , state_list )
    pick_priority_set   ( state_element, state_list )
    priority_set        ( state_list )
    append_best_goal    ( state_list, state_element )

  below_limit           ( status_list )
  changed               ( status_list )
  update                ( state_list )
  scan_panel            ( segment_list, state_list )
  segment_panel         ( segment_list )

/* utilities
*/
  deleteitems           ( state_list, state_list, state_list )
  delete                  (    state_element,    state_list,
state_list )
  union                 ( state_list, state_list, state_list )
  append                ( op_list, op_list, op_list )
  difference            ( state_list, state_list, state_list )
  subset                ( state_list, state_list )
  member                ( statc_clement, state_list )
  member2               ( state_element, state_list )
```

```
  status_member        ( component_status, status_list )

/* test cases
*/
  test1                ( op_list,state_list )
  test2                ( op_list,state_list )
  test3                ( op_list,state_list )
  test4                ( op_list,state_list )
```

```
/*********************** CLAUSES ***********************/
```

uses

```
/*  Problem_solver is the top level predicate from which all
other   predicates are called.  The goals are initialized to
empty.  The      guages are first searched for components  in
need of some type of   response. Then the caution lights are
searched.  Once the goals have    been asserted means_ends is
called  to  present the operators necessary   to  solve  the
problem.
*/
  problem_solver( State,Oplist,Goalstate ):-
                  asserta(temp_goals([])),
                  check_guages(State ),
                  check_lights(State, Goal_list),!,
                  means_ends(  State,   Goal_list,   Oplist,
                               Goalstate, 1 ),!.

/*  This recursive procedure has two rules.  The first is  a
single  basis     step,  which in effect says stop with  any
state that includes all the    goal facts.  The second  rule
is the induction step which has two re-   cursive calls: the
first   for   the   preconditions,   the   second   for   the
postconditions.  A  list  of  facts  is  computed  that  are
different     between the current state and  the  goal.  The
recommended  facts are   searched in order to find one whose
goal is a subset of the goal   list. If so then retrieve the
preconditions of the operator,  and   recursively call means
ends to resolve the differences.  Once the      preconditions
have   been   satisfied  the deletepostcondition  facts    are
deleted from the final state resulting from the precondition
recursion. Then the addpostcondition facts are retrieved and
added    to the state.  This determines the state after  the
operator  application.    This process is  done  recursively
until  all  preconditions have been   satisfied.  The  final
operator  list  for the whole problem  is  the    appending
together of the precondition-recursion operator list,    the
recommended operator, and the postcondition operator list.
*/
  means_ends( State, Goal_list, [], State, N ) :-
              difference( Goal_list, State, [] ),
              write("operator preconditions have been met"),
              nl,nl.

  means_ends( State, Goal_list, Oplist, Goalstate, N ) :-
              difference( Goal_list, State, D ),
              write("Goal list is: ",Goal_list),nl,nl,
              write("difference:",D),nl,nl,
              recommended( State,  Dsub, Operator ),
              subset( Dsub, D ),
```

```
                 write("************************************"),
                 nl,nl,
                 write("current operator: ",Operator),
                 write("    depth is: ", N ),nl,nl,
                 write("************************************"),
                 nl,nl,
                 precondition( State, Operator, Prelist ),
                 N2 = N + 1,
                 means_ends( State, Prelist, Preoplist,
                             Prestate, N2 ),
                 deletepostcondition( State, Operator,
                                      Deletepostlist ),
                 deleteitems( Deletepostlist, Prestate,
                              Prestate2 ),
                 write("items being deleted are: ",
                       Deletepostlist),nl,nl,
                 addpostcondition( Operator, Addpostlist ),
                 write("items being added are: ",
                       Addpostlist),nl,nl,
                 union( Addpostlist, Prestate2, Postlist ),
                 means_ends(Postlist, Goal_list, Postoplist,
                            Goalstate, N2 ),
                 append( Preoplist, [ Operator | Postoplist ],
                        Oplist ),!.

/*  'create goals' is  a control structure that  provides  a
way  to  generate   goals from within  the  program.  Before
goals  are  appended to the goal  list,  they  are  checked
against  other  goals in the same class to   eliminate  the
appending of duplicates. Each class is a list of prioritized
state elements. If a goal of a  higher priority has already
been appended then the goal under con-  sideration will not
be appended.  create_goals looks at the first of the  goals
to  be  considered.  If  a duplicate already exists  in  the
present   goal list the ck_duplicate fails and create_goals
is called again to consider the next goal.  If there are  no
duplicates,  append_best_goal  checks    to see if  another
goal  of  a  higher  priority  within  the  same  class  has
already  been asserted.If no higher goal has  been  asserted
then   append_best_goal  adds the new goal to the new  goal
list.
*/
  create_goals(  [] ):- !.   /* succedes when no  goals  are
                                specified. */

  create_goals( [G1|Rest] ) :-
                 temp_goals(Current_goals ),
                 ck_duplicate( G1, Current_goals ),
                 pick_priority_set(G1, Set ),
                 append_best_goal(Set,G1),
                 create_goals( Rest ),!.
```

```
  create_goals( [G1|Rest] ) :- !, create_goals( Rest ).

  ck_duplicate( X, L ) :- not( member ( X, L )).

/*  before  a  goal  is asserted into the goal  list  it  is
checked against   other goals in its priority set to see  if
a goal of a higher priority   has already been asserted. Any
goal which is not explicitly listed   in a priority set fact
is assumed to be the only member in it's   class.
*/
  pick_priority_set( X, C )  :- priority_set( C ),
                                member( X, C ).
  pick_priority_set( X, [X] ). /* default for singleton sets
                               */



  /*  If  the highest priority item matches the  goal   then
append  it.     check  to see if a higher proirity goal  has
been  asserted.  The procedure stops at the higher  priority
goal if already asserted
*/
 append_best_goal( [SingleGoal|RestSet],SingleGoal ) :-
                      temp_goals(Current_goals),
          update([SingleGoal|Current_goals]),!.
                    /* assertion */
 append_best_goal( [S1|RestSet],Single_goal ) :-
                    /* no assertion */
                      temp_goals(Current_goals),
                      ck_duplicate( S1,Current_goals ),
                      append_best_goal( RestSet,Single_goal ).

 update( New_goals ) :-
        retract(temp_goals(_)),
        asserta(temp_goals(New_goals)),!.

   difference([],S,[]).
   difference( [ P | G ], S, G2 ) :-
               member( P, S ), !,
               difference( G, S, G2 ).
   difference( [ P | G ], S, [ P | G2 ] ) :-
               difference( G, S, G2 ).

   subset( [], L ).
   subset( [ X | L ], L2 ) :-
           member( X, L2 ),
           subset( L, L2 ).

   member( X, [ X | L ] ) :- !.
   member( X, [ Y | L ] ) :-  member( X, L ).
```

```
  member2( X, [ X | L ] ).
  member2( X, [ Y | L ] ) :-  member2( X, L ).

  status_member( X, [ X | L ] ) :- !.
  status_member( X, [ Y | L ] ) :-  status_member( X, L ).

  append( [], L, L ).
  append( [ X | L ], L2, [ X | L3 ] ) :-
          append( L, L2, L3 ).

  union( [], L, L ).
  union( [ X | L1 ], L2, L3 ) :- member( X, L2 ), !,
                                 union( L1, L2, L3 ).
  union( [ X | L1 ], L2, [ X | L3 ] ) :-
         union( L1, L2, L3 ).

 deleteitems( [], L, L ).
 deleteitems( [ X | L ], L2, L3 ) :-
                           delete( X, L2, L4 ),
                           deleteitems( L, L4, L3 ).

 delete( X, [], [] ).
 delete( X, [ X | L ], M ) :- !, delete( X, L, M ).
 delete( X,[ Y | L ], [ Y | M ] ) :- delete( X, L, M ).


/*  The performance guages are the primary area of  interest
and involve the temperatures, pressures, torque,and RPM's of
various components.
*/
 check_guages( State ) :-
               check_revs( State ),
               check_pressures( State ),
               check_temps( State ),
               check_torque( State ).

/*   The  rules   check_revs,check_pressures,check_temps,and
check_torque  are very similar.  Each guage is checked for a
change  in value.  A change in value requires a response  of
some sort.  The goal is to respond to the guage or light  in
question.
*/
 check_revs( State ) :-
             member2(revs(Guage, Value ), State ),
             changed(Up_or_Down),
             status_member(Value,Up_or_Down),
             write("revs(",Guage,",     ",Value,")"),nl,nl,
             create_goals([revs(Guage,respond)]),fail.
 check_revs( State ).
```

```
  check_pressures( State ) :-
              member2(oil_press(Guage, Value ), State ),
              changed(Up_or_Down),
              status_member(Value,Up_or_Down),
              write("oil_press(",Guage,", ",Value,")" ),
              nl,nl,
              create_goals([oil_press(Guage,respond)]),fail.
  check_pressures( State ).

  check_temps( State ) :-
              member2(oil_temp(Guage, Value ), State ),
              changed(Up_or_Down),
              status_member(Value,Up_or_Down),
              write("oil_temp(",Guage,", ",Value,")" ),
              nl,nl,
              create_goals([oil_temp(Guage,respond)]),fail.
  check_temps( State ).

  check_torque( State ) :-
              member2(torque(Guage, Value ), State ),
              changed(Up_or_Down),
              status_member(Value,Up_or_Down),
              write("torque(",Guage,", ",Value,")" ),
              nl,nl,
     create_goals([torque(Guage,respond)]),fail.
  check_torque( State ).


/*  check_lights  goes  through all the caution  lights  and
accumulates  all the goals necessary to correct the  present
state with respect to the caution panel.
*/
  check_lights( State,Goal_list ) :-
                segment_panel(Cluster),
                scan_panel(Cluster,State).
  check_lights( State, Goal_list ) :-
                temp_goals(Goal_list),
                retract(temp_goals(_)).

/* scan_panel calls each caution light in the panel
*/
  scan_panel([],State):-!,fail.
  scan_panel([Seg|Tail],State):-
            caution_light(State,Seg),!,
            scan_panel(Tail,State).
  scan_panel([Seg|Tail],State):-scan_panel(Tail,State).
```

```
/************CAUTION AND WARNING LIGHTS *************/

  segment_panel([ engl_oil_press,    eng2_oil_press,
                  engl_chip_detr,    eng2_chip_detr,
                  engl_fuel_filter,  eng2_fuel_filter,
                  dc_gen_1,          dc_gen_2,
                  xmsn_chip_detr,    c_box_chip_detr,
                  temp_press_90,     temp_press_42,
                  chip_detr_90,      chip_detr_42,
                  xmsn_oil_hot,      xmsn_oil_press,
                  c_box_oil_press,   c_box_oil_hot ,
                  hyd_press_1,       hyd_press_2,
                  hyd_temp_1,        hyd_temp_2,
                  ac_main,           ac_stby ,
                  engl_gov_man,      eng2_gov_man,
                  fire_1_pull,       fire_2_pull,
                  rpm_rotor_low,     rpm_rotor_high,
                  rpm_ngl,           rpm_ng2,
                  xmsn_oil_byp ]).

/*  The master_caution light is illuminated if there  exists
any caution light that is on
*/
  master_caution( State, Goal_list ) :-
                  caution_light( State, _ ).

/*  Listed below are the rules for each caution light.  Each
caution light has its own goals to assert and in some  cases
secondary  indications  are  necessary  to  choose  the
appropriate  goals.  All sensor information is assumed  true
and  is not questioned.  This assumption  eliminates  cross-
checking requirements.
*/
  caution_light( State,  engl_oil_press):-
             member(oil_press( engl, X ), State ),
             below_limit(L), status_member(X,L),
             write( " Caution light: engl_oil_press" ),nl,
             create_goals([ off(engl), land(pract) ] ).

  caution_light( State,  engl_chip_detr ):-
             member(metal_particles(engl), State ),
             write( " Caution light: engl_chip_detr " ),nl,
             secondary(State, engl_chip_detr).

  caution_light( State,  eng2_oil_press ):-
             member(oil_press( eng2, X ), State ),
             below_limit(L), status_member(X,L),
             write( " Caution light: eng2_oil_press" ),nl,
             create_goals([ off(eng2), land(pract) ] ).

  caution_light( State,  eng2_chip_detr ):-
```

```
          member(metal_particles(eng2), State ),
          write( " Caution light: eng2_chip_detr " ),nl,
          secondary(State, eng2_chip_detr).

caution_light( State,  eng1_fuel_filter ):-
          member(fuel_obstruction(eng1), State),
          write( " Caution light: eng1_fuel_filter "),nl,
          create_goals([prepare_for_failure(eng1),
                        land(pract) ] ).

caution_light( State,  eng2_fuel_filter ):-
          member(fuel_obstruction(eng2), State),
          write( " Caution light: eng2_fuel_filter "),nl,
          create_goals([prepare_for_failure(eng2),
                        land(pract) ] ).

caution_light( State,  dc_gen_1 ):-
          member(off(gen1), State ),
          write( " Caution light: dc_gen_1 " ),nl,
          create_goals([] ).

caution_light( State,  dc_gen_1 ):-
             member(failed(gen1), State),
             write( " Caution light: dc_gen_1 " ),nl,
             create_goals([ off(gen1) ] ).

caution_light( State,  dc_gen_2 ):-
             member(off(gen2), State ),
             write( " Caution light: dc_gen_2 " ),nl,
             create_goals([] ).

caution_light( State,  dc_gen_2 ):-
             member(failed(gen2), State),
             write( " Caution light: dc_gen_2 " ),nl,
             create_goals([ off(gen2) ] ).

caution_light( State,  xmsn_chip_detr ):-
          member(metal_particles( xmsn ), State ),
          write( " Caution light: xmsn_chip_detr " ),nl,
          secondary(State, xmsn_chip_detr).

caution_light( State,  c_box_chip_detr ):-
          member(metal_particles( c_box ), State ),
          write( " Caution light: c_box_chip_detr " ),nl,
          create_goals([power(decrease),land(pos) ] ).

caution_light( State,  temp_press_90 ):-
          member(oil_temp( gear_box_90, high ), State ),
          write( " Caution light: temp_press_90 " ),nl,
          create_goals([ land(pos) ] ).
```

```
caution_light( State,  temp_press_90 ):-
        member(oil_press( gear_box_90, X ), State ),
        below_limit(L), status_member(X,L),
        write( " Caution light: temp_press_90 " ),nl,
        create_goals([ land(pos) ] ).

caution_light( State,  temp_press_42 ):-
        member(oil_temp( gear_box_42, high ), State ),
        write( " Caution light: temp_press_42 " ),nl,
        create_goals([ land(pos) ] ).

caution_light( State,  temp_press_42 ):-
        member(oil_press( gear_box_42, X ), State ),
        below_limit(L), status_member(X,L),
        write( " Caution light: temp_press_42 " ),nl,
        create_goals([ land(pos) ] ).

caution_light( State,  chip_detr_90 ):-
        member(metal_particles(gear_box_90),State ),
        write( " Caution light: chip_detr_90 " ),nl,
        create_goals([power(decrease),land(pract)]).

caution_light( State,  chip_detr_42 ):-
       member(metal_particles(gear_box_42 ),State ),
       write( " Caution light: chip_detr_42 " ),nl,
       create_goals([power(decrease),land(pract)] ).

caution_light( State,  xmsn_oil_hot ):-
        member(oil_temp( xmsn, high ), State ),
        write( " Caution light: xmsn_oil_hot " ),nl,
        create_goals([ power(decrease),
                       prepare_for_failure(xmsn),
                       land(pos) ] ).

caution_light( State,  xmsn_oil_press ):-
        member(oil_press( xmsn, X ), State ),
        below_limit(L), status_member(X,L),
        write( " Caution light: xmsn_oil_press " ),nl,
        create_goals([ power(decrease),
                       prepare_for_failure(xmsn),
                       land(pos) ] ).

caution_light( State,  c_box_oil_press ):-
        member(oil_press( c_box, high ), State ),
        write( " Caution light: c_box_oil_press " ),nl,
        create_goals([ power(decrease), land(pos)  ] ).

caution_light( State,  c_box_oil_hot ):-
        member(oil_temp( c_box, high ), State ),
        write( " Caution light: c_box_oil_hot " ),nl,
        create_goals([ power(decrease), land(pos) ] ).
```

```
caution_light( State,  hyd_press_1 ):-
          member(oil_press( hyd_sys_1, X ), State ),
          below_limit(L), status_member(X,L),
          write( " Caution light: hyd_press_1 " ),nl,
          create_goals([ off(hyd_sys_1),land(pos) ] ).

caution_light( State,  hyd_press_2 ):-
          member(oil_press( hyd_sys_2, X ), State ),
          below_limit(L), status_member(X,L),
          write( " Caution light: hyd_press_2 " ),nl,
          create_goals([  off(hyd_sys_2),land(pos) ] ).

caution_light( State,  hyd_temp_1 ):-
          member(oil_temp( hyd_sys_1, high ), State ),
          write( " Caution light: hyd_temp_1 " ),nl,
          create_goals([ off(hyd_sys_1),land(pos) ] ).

caution_light( State,  hyd_temp_2 ):-
          member(oil_temp( hyd_sys_2, high ), State ),
          write( " Caution light: hyd_temp_2 " ),nl,
          create_goals([ off(hyd_sys_2),land(pos) ] ).

caution_light( State,  xmsn_oil_byp ):-
          member(oil_bypassing_cooler, State ),
          write( " Caution light: xmsn_oil_byp " ),nl,
          create_goals([power(decrease),
                        prepare_for_failure(xmsn),
                        land(pos) ] ).

caution_light( State,  ac_main ):-
          member(failed( main_inverter ), State ),
          write( " Caution light: ac_main " ),nl,
          create_goals([] ). /* not implemented */

caution_light( State,  ac_stby ):-
          member(failed( standby_inverter ), State ),
          write( " Caution light: ac_stby " ),nl,
          create_goals([] ). /* not implemented */

caution_light( State,  eng1_gov_man ):-
          member(manual( gov1), State ),
          write( " Caution light: eng1_gov_man " ),nl,
          create_goals([] )./* no goal required */

caution_light( State,  eng2_gov_man ):-
          member(manual( gov2), State ),
          write( " Caution light: eng2_gov_man " ),nl,
          create_goals([] )./* no goal required */

caution_light( State,  fire_1_pull ):-
```

```
            member(fire( eng1 ), State ),
            write( " Caution light: fire_1_pull " ),nl,
            create_goals([] ).

  caution_light( State,   fire_2_pull ):-
            member(fire( eng2 ), State ),
            write( " Caution light: fire_2_pull " ),nl,
            create_goals([] ).

  caution_light( State,  rpm_ng1 ):-
            member(revs( ng1, X ), State ),
            below_limit(L), status_member(X,L),
            write( " Caution light: rpm_ng1 " ),nl,
            create_goals([ revs(ng1,respond) ] ).

  caution_light( State,  rpm_ng2 ):-
            member(revs( ng2, X ), State ),
            below_limit(L), status_member(X,L),
            write( " Caution light: rpm_ng2 " ),nl,
            create_goals([ revs(ng2,respond) ] ).

  caution_light( State,  rpm_rotor_low ):-
            member(revs( rotor, low ), State ),
            write( " Caution light: rpm_rotor_low " ),nl,
            create_goals([ revs(rotor,respond) ] ).

  caution_light( State,  rpm_rotor_high ):-
            member(revs( rotor, high ), State ),
            write( " Caution light: rpm_rotor_high " ),nl,
            create_goals([ revs(rotor,respond) ] ).


/*  secondary state-requirements for certain caution  lights
are expressed below.
*/
  secondary(State, eng1_chip_detr) :-
           member(oil_press(eng1, X ), State ),
           below_limit(L), status_member(X,L),
           create_goals([ off(eng1),land(pract)] ).
  secondary(State, eng1_chip_detr) :-
           member(oil_temp(eng1, high), State ),
           create_goals([ off(eng1),land(pract)] ).
  secondary(State, eng1_chip_detr) :-
           create_goals([ idle(throttle1),land(pract)] ).

  secondary(State, eng2_chip_detr) :-
           member(oil_press(eng2, X ), State ),
           below_limit(L), status_member(X,L),
           create_goals([ off(eng2),land(pract)] ).
  secondary(State, eng2_chip_detr) :-
           member(oil_temp(eng2, high), State ),
```

```
         create_goals([ off(eng2),land(pract)] ).
 secondary(State, eng2_chip_detr) :-
         create_goals([ idle(throttle2),land(pract)] ).

 secondary(State, xmsn_chip_detr) :-
         member(oil_temp(xmsn,high), State),
         create_goals([prepare_for_failure(xmsn),
                       land(pos)] ).
 secondary(State, xmsn_chip_detr) :-
         member(oil_press(xmsn,X ), State),
         below_limit(L), status_member(X,L),
         create_goals([prepare_for_failure(xmsn),
                       land(pos)] ).
 secondary(State, xmsn_chip_detr) :-
         create_goals([ land(pos)] ).

changed([erratic,increase,decrease]).

below_limit([none,low]).
```

```
/*<<<<<<<<<<<<<<<<< recommended operators >>>>>>>>>>>>>>>>>*/

/* The recommended facts gives the operator to apply to
achieve the goals listed. The order in which these facts are
listed is crutial to the over all performance of the
planner. Those opeators that are the most difficult to
satisty come before those that are eaiser. Operators are
also listed in order of urgency. A given goal may have more
than one operator that is suitable. These operators are
ordered in the same manner as above.
*/
  recommended( State, [revs(ng2,respond),revs(ng1,respond),
                       off(gen1),off(gen2),
                       off(eng1),off(eng2),land(pract) ],
              confirm_dual_eng_failure ).

  recommended( State, [revs(rotor,respond)],
          confirm_eng2_failure).
  recommended( State, [revs(rotor,respond)],
         confirm_eng1_failure).
  recommended( State, [revs(rotor,respond)],
         gov1_to_manual).
  recommended( State, [revs(rotor,respond)],
         gov2_to_manual).

  recommended( State, [torque(xmsn,respond)],
          check_ng1_underspeed).
  recommended( State, [torque(xmsn,respond)],
          check_ng2_underspeed).

  recommended( State, [torque(eng1,respond)],
          confirm_nf1_failure).
  recommended( State, [torque(eng1,respond)],
          check_ng1_overspeed).
  recommended( State, [torque(eng1,respond)],
          check_ng1_underspeed).

  recommended( State, [torque(eng2,respond)],
          confirm_nf2_failure).
  recommended( State, [torque(eng2,respond)],
          check_ng2_overspeed).
  recommended( State, [torque(eng2,respond)],
          check_ng2_underspeed).

  recommended( State, [revs(ng1,respond)],
          gov1_to_manual).

  recommended( State, [revs(nf1,respond)],
          gov1_to_manual).

  recommended( State, [revs(ng2,respond)],
```

```
        gov2_to_manual).

recommended( State, [revs(nf2,respond)],
        gov2_to_manual).

recommended( State, [land(auto)], autorotation).

recommended( State, [failed(eng2)],
        confirm_eng2_failure).
recommended( State, [failed(eng1)],
        confirm_eng1_failure).

recommended( State, [failed(gov1)],confirm_nf1_failure).
recommended( State, [failed(gov1)],check_ng1_overspeed).
recommended( State, [failed(gov1)],check_ng1_underspeed).
recommended( State, [failed(gov2)],confirm_nf2_failure).
recommended( State, [failed(gov2)],check_ng2_overspeed).
recommended( State,  [failed(gov2)],check_ng2_underspeed).

recommended( State, [itt(eng1,respond)],gov1_to_manual).
recommended( State, [itt(eng2,respond)],  gov2_to_manual).

recommended( State, [failed(gen1)],check_gen1_failure).
recommended( State, [failed(gen2)], check_gen2_failure).

recommended( State, [engaged(scas)],        engage_scas ).

recommended( State, [automatic(gov2)],gov2_to_automatic).
recommended( State, [automatic(gov1)],gov1_to_automatic).

recommended( State, [idle(throttle1)],throttle1_idle ).
recommended( State, [idle(throttle2)],throttle2_idle ).

recommended( State, [off(eng1)],                eng1_off ).
recommended( State, [off(eng2)],                eng2_off ).
recommended( State, [off(ecu)],                 ecu_off ).
recommended( State, [off(gen1)],                gen1_off ).
recommended( State,  [off(gen2)],                 gen2_off  ).
recommended( State, [off(rain_rmv)],    rain_rmv_off ).
recommended( State, [off(master_arm)],  master_arm_off ).

recommended( State, [land(pos)],pos_landing_outranked ).
recommended( State, [land(pos)],       pos_landing1 ).
recommended( State, [land(pos)],       pos_landing2 ).

recommended( State, [land(pract)],
            pract_landing_outranked1 ).
recommended( State, [land(pract)],     pract_landing ).

recommended( State, [land(normal)],    normal_approach ).
recommended( State, [land(slope)],     slope_landing ).
```

```
recommended( State, [land(steep)],     steep_approach ).
recommended( State, [land(high_speed)],
             high_speed_approach ).
recommended( State,  [land(max_weight)],
             max_weight_landing ).
recommended( State,  [land(sliding)],   sliding_landing ).
```

```
/*<<<<<<<<<<<<<<<<<<<< preconditions >>>>>>>>>>>>>>>>>>>>*/

/*  The  preconditions for the application of operators  are
defined in these facts
*/
  precondition( State, gen1_off ,[]).

  precondition( State, gen2_off ,[]).

  precondition( State, eng1_off ,[]).

  precondition( State, eng2_off ,[]).

  precondition( State, check_gen1_failure ,[on(gen1),
               ammeter(eng1,none)]).

  precondition( State, check_gen2_failure ,[on(gen2),
               ammeter(eng2,none)]).

  precondition( State, gov1_to_manual,
             [automatic(gov1), automatic(gov2),
             failed(gov1), idle(throttle1)]).

  precondition( State, gov1_to_automatic,
             [idle(throttle1),manual(gov1)]).

  precondition( State, confirm_nf1_failure,
             [revs(ng1,erratic),itt(eng1,erratic),
             revs(nf1,erratic),
             torque(eng1,erratic), automatic(gov1)]).

  precondition( State, check_ng1_underspeed,
             [revs(ng1,decrease),revs(nf1,decrease),
             torque(eng1,decrease),automatic(gov1)]).

  precondition( State, check_ng1_overspeed,
             [revs(ng1,increase),revs(nf1,increase),
             torque(eng1,increase),automatic(gov1)]).


  precondition( State, gov2_to_manual,
             [automatic(gov2), automatic(gov1),
             failed(gov2), idle(throttle2)]).

  precondition( State, gov2_to_automatic,
             [idle(throttle2),manual(gov2)]).

  precondition( State, confirm_nf2_failure,
             [revs(ng2,erratic),itt(eng2,erratic),
             revs(nf2,erratic),torque(eng2,erratic),
             automatic(gov2)]).
```

```
precondition( State, check_ng2_underspeed,
            [revs(ng2,decrease),revs(nf2,decrease),
            torque(eng2,decrease),automatic(gov2)]).

precondition( State, check_ng2_overspeed,
            [revs(ng2,increase),revs(nf2,increase),
            torque(eng2,increase),automatic(gov2)]).

precondition( State, autorotation, [failed(eng1),
              failed(eng2)]).

precondition( State, high_speed_approach,[ fire(_)]).

precondition( State, max_weight_landing,
              [ gross_weight(heavy),
                landing_zone(clear)]).

precondition( State, sliding_landing, [power(low),
              landing_zone(clear)]).

precondition( State, pos_landing_outranked,
              [land(auto)]).

precondition( State, pos_landing1,
              [failed(eng1)]).

precondition( State, pos_landing2,
              [failed(eng2)]).

precondition( State, pract_landing,
              [landing_zone(clear)]).

precondition( State, pract_landing_outranked1,
              [land(pos)]).

precondition( State, normal_approach,
              [ revs( rotor, ok ), off( master_caution ),
              engaged( scas ),off(ecu),off(rain_rmv),
              off( master_arm ), landing_zone(clear)]).

precondition( State, slope_landing,
             [ revs( rotor,  ok ), off( master_caution ),
             engaged( scas ),off( ecu ),off( rain_rmv  ),
             off(master_arm ),
             landing_zone(slope),gross_weight(moderate)]).

precondition( State, confirm_eng1_failure,
      [torque(eng1,none), torque(xmsn,low), revs(ng1,none),
      revs(nf1,none), oil_press(eng1,none), failed(gen1),
      itt(eng1,none), left_yaw, on(fuel1),full(throttle1),
```

```
     revs(rotor,low)]).

precondition( State, confirm_eng2_failure,
     [torque(eng2,none), torque(xmsn,low),revs(ng2,none),
     revs(nf2,none), oil_press(eng2,none), failed(gen2),
     itt(eng2,none), left_yaw, on(fuel2), full(throttle2),
     revs(rotor,low)]).

precondition( State, confirm_dual_eng_failure,
     [ oil_press(eng1,none), oil_press(eng2,none),
     revs(rotor,low), full(throttle1), full(throttle2),
     on(fuel1), on(fuel2), torque(eng1,none),
     torque(eng2,none), torque(xmsn,low), revs(ng1,none),
     revs(ng2,none), revs(nf1,none), revs(nf2,none),
     failed(gen1), failed(gen2),off(scas),itt(eng1,none),
     itt(eng2,none), left_yaw ]).

precondition( State, engage_scas,[] ).
precondition( State, ecu_off,[] ).
precondition( State, rain_rmv_off,[]).
precondition( State, master_arm_off,[]).
precondition( State, throttle1_idle,[]).
precondition( State, throttle2_idle,[]).
```

```
/*<<<<<<<<<<<<<<<<< deletepostconditions >>>>>>>>>>>>>>>>>*/

/*  The  respective  state elements to be deleted  from  the
state  description,  for each operator are defined in  these
facts.
*/

  deletepostcondition( State, gen1_off, [on(gen1)] ).

  deletepostcondition( State, gen2_off, [on(gen2)] ).

  deletepostcondition( State, eng1_off, [on(eng1)] ).

  deletepostcondition( State, eng2_off, [on(eng2)] ).

  deletepostcondition( State,check_gen1_failure,[on(gen1)]).

  deletepostcondition( State,check_gen2_failure,[on(gen2)]).

  deletepostcondition( State, confirm_eng1_failure,
                       [on(ecu)] ).

  deletepostcondition( State, confirm_eng2_failure,
                       [on(ecu)] ).

  deletepostcondition( State, confirm_dual_eng_failure,
                      [on(ecu)] ).

  deletepostcondition( State,gov1_to_manual,
                      [automatic(gov1), idle(throttle1),
                      revs(ng1,_), itt(eng1,_),
                      revs(nf1,_),revs(rotor,_),
                      torque(eng1,_)]).

  deletepostcondition( State,gov1_to_automatic,
                      [manual(gov1),manual(throttle1)]).

  deletepostcondition( State,confirm_nf1_failure,[]).

  deletepostcondition( State,check_ng1_underspeed,[]).

  deletepostcondition( State,check_ng1_overspeed,[]).

  deletepostcondition( State,gov2_to_manual,
             [automatic(gov2),idle(throttle2),revs(ng2,_),
             itt(eng2,_),revs(nf2,_),revs(rotor,_),
             torque(eng2,_)]).

  deletepostcondition( State,gov2_to_automatic,
                      [manual(gov2),manual(throttle2)]).
```

```
deletepostcondition( State,confirm_nf2_failure,[]).

deletepostcondition( State,check_ng2_underspeed,[]).

deletepostcondition( State,check_ng2_overspeed,[]).

deletepostcondition( State, throttle1_idle,
                     [full(throttle1)]).

deletepostcondition( State, throttle2_idle,
                     [full(throttle2)]).

deletepostcondition( State, ecu_off, [on(ecu)]).

deletepostcondition( State, engage_scas, [off(scas)]).

deletepostcondition( State, rain_rmv_off, [on(rain_rmv)]).

deletepostcondition( State, master_arm_off,
                     [on(master_arm)]).

deletepostcondition( State, normal_approach, [altitude(_),
                     airspeed(_)]).

deletepostcondition( State, slope_landing,   [altitude(_),
                     airspeed(_)]).

deletepostcondition( State, steep_approach,  [altitude(_),
                     airspeed(_)]).

deletepostcondition( State, high_speed_approach,
                  [altitude(_), airspeed(_)]).

deletepostcondition( State, max_weight_landing,
                  [altitude(_), airspeed(_)]).

deletepostcondition( State, sliding_landing,
                  [altitude(_), airspeed(_)]).

deletepostcondition( State, pract_landing_outranked1,
                  []).

deletepostcondition( State, pract_landing,
                  [altitude(_), airspeed(_)]).

deletepostcondition( State, pos_landing_outranked,
                  []).

deletepostcondition( State, pos_landing1,
                  [altitude(_), airspeed(_)]).
```

```
deletepostcondition( State, pos_landing2,
                    [altitude(_), airspeed(_)]).

deletepostcondition( State, autorotation,
          [full(throttle1), full(throttle2),airspeed(_),
          altitude(_), revs(rotor,_)]).
```

```
/*<<<<<<<<<<<<<<<<< addpostconditions >>>>>>>>>>>>>>>>>>*/

/*  The respective state elements to be added to  the  state
description, for each operator are defined in these facts.
*/

  addpostcondition( check_gen1_failure,
                  [ off(gen1),failed(gen1)]).

  addpostcondition( check_gen2_failure,
                  [ off(gen2),failed(gen2)]).

  addpostcondition( gen1_off,
                  [ off(gen1) ]).

  addpostcondition( gen2_off,
                  [ off(gen2) ]).

  addpostcondition( eng1_off,
                  [ off(eng1) ]).

  addpostcondition( eng2_off,
                  [ off(eng2) ]).


  addpostcondition( autorotation,
               [idle(throttle1), idle(throttle2),
               airspeed(0), altitude(0),revs(rotor,respond),
               land(auto)]).

  addpostcondition( confirm_eng1_failure,
                  [ off(ecu),failed(eng1),revs(ng1,respond),
                  revs(nf1,respond),    revs(rotor,respond),
                  itt(eng1,respond), torque(eng1,respond),
                  torque(xmsn,respond)]).

  addpostcondition( confirm_eng2_failure,
         [ off(ecu),failed(eng2),revs(ng2,respond),
         revs(nf2,respond), revs(rotor,respond),
         itt(eng2,respond), torque(eng2,respond),
         torque(xmsn,respond)]).

  addpostcondition( confirm_dual_eng_failure,
         [ off(ecu),failed(eng1),failed(eng2),off(gen1),
         off(gen2), off(eng1), off(eng2), revs(ng1,respond),
         revs(nf1,respond), revs(rotor,respond),
         itt(eng1,respond), itt(eng2,respond),
         revs(ng2,respond), revs(nf2,respond)]).


  addpostcondition(gov1_to_manual,
```

```
                 [manual(gov1), manual(throttle1),
                 revs(ng1,ok), itt(eng1,ok), revs(nf1,ok),
                 revs(rotor,ok), torque(eng1,ok),
                 torque(eng1,respond), revs(ng1,respond),
                 revs(nf1,respond),revs(rotor,respond),
                 itt(eng1,respond)]).

addpostcondition(gov1_to_automatic, [automatic(gov1)]).

addpostcondition(confirm_nf1_failure,[revs(ng1,respond),
                 revs(nf1,respond), failed(gov1),
                 torque(eng1,respond)]).

addpostcondition(check_ng1_underspeed,[revs(ng1,respond),
                 revs(nf1,respond), failed(gov1),
                 torque(eng1,respond)]).

addpostcondition(check_ng1_overspeed,[revs(ng1,respond),
                 revs(nf1,respond), failed(gov1),
                 torque(eng1,respond)]).

addpostcondition(gov2_to_manual,
       [manual(gov2), manual(throttle2), revs(ng2,ok),
       itt(eng2,ok),revs(nf2,ok),revs(rotor,ok),
       torque(eng2,ok),torque(eng2,respond),
       revs(ng2,respond), revs(nf2,respond),
       revs(rotor,respond),itt(eng2,respond)]).

addpostcondition(gov2_to_automatic, [automatic(gov2)]).

addpostcondition(confirm_nf2_failure,[revs(ng2,respond),
       revs(nf2,respond), failed(gov2),
       torque(eng2,respond)]).

addpostcondition(check_ng2_underspeed,[revs(ng2,respond),
                 revs(nf2,respond), failed(gov2),
                 torque(eng2,respond)]).

addpostcondition(check_ng2_overspeed,[revs(ng2,respond),
                 revs(nf2,respond), failed(gov2),
                 torque(eng2,respond)]).

addpostcondition( throttle1_idle, [ idle( throttle1)]).

addpostcondition( throttle2_idle, [ idle( throttle2)]).

addpostcondition( ecu_off, [off(ecu)]).

addpostcondition( engage_scas, [engaged(scas)]).

addpostcondition( rain_rmv_off, [off(rain_rmv)]).
```

```
addpostcondition( master_arm_off, [off(master_arm)]).

addpostcondition( normal_approach,
                [land(normal), altitude(0), airspeed(0)]).

addpostcondition( slope_landing,
                [land(slope), altitude(0), airspeed(0)]).

addpostcondition( steep_approach,
                [land(steep), altitude(0), airspeed(0)]).

addpostcondition( high_speed_approach,
            [land(high_speed), altitude(0), airspeed(0)]).

addpostcondition( max_weight_landing,
            [land(max_weight), altitude(0), airspeed(0)]).

addpostcondition( sliding_landing,
            [land(sliding), altitude(0), airspeed(0)]).

addpostcondition( pos_landing_outranked,
                  [land(pos)]).

addpostcondition( pos_landing1,
                [land(pos), altitude(0), airspeed(0)]).

addpostcondition( pos_landing2,
                [land(pos),   altitude(0),  airspeed(0)]).

addpostcondition( pract_landing,
                [land(pract), altitude(0), airspeed(0)]).

addpostcondition( pract_landing_outranked1,
                  [land(pract)]).
```

```
/*<<<<<<<<<<<<<<<<<<<<<< PRIORITY SETS >>>>>>>>>>>>>>>>>>>>>>*/

/* These sets define the relative priority among its
elements. If an element of a higher priority has been
asserted as a goal no lower priority elements will be
asserted.
*/
          priority_set([ failed(eng1), off(eng1)]).

  priority_set([ failed(eng2), off(eng2)]).

  priority_set([land(pos), land(pract)]).

  priority_set([land(slope), land(high_speed),
             land(max_weight), land(sliding), land(steep),
             land(normal)]).

  priority_set([power(none), power(low), power(decrease),
             power(increase)]).

  priority_set([power(none), power(low), power(ok)]).
```

```
/*<<<<<<<<<<<<<<<<<<<< test cases >>>>>>>>>>>>>>>>>>>>*/

                /*    test1 normal state */

test1(Oplist,Goalstate) :-   problem_solver

(  [  revs(rotor,ok),  full(fuel),  engaged(scas),  on(ecu),
off(rain_rmv),     off(master_arm),     landing_zone(clear),
full(throttle1),    full(throttle2),    off(master_caution),
automatic(gov1),   automatic(gov2),   on(fuel1),  on(fuel2),
normal(fuel_press),      torque(eng1,ok),     torque(eng2,ok),
torque(xmsn,ok),  revs(ng1,ok),  revs(ng2,ok), revs(nf1,ok),
revs(nf2,ok),       oil_temp(eng1,ok),      oil_temp(eng2,ok),
oil_press(eng1,ok),  oil_press(eng2,ok), on(gen1), on(gen2),
oil_temp(c_box,ok),  oil_press(c_box,ok), oil_temp(xmsn,ok),
oil_press(xmsn,ok),    ammeter(gen1,ok),   ammeter(gen2,ok),
fuel_press(ok),           itt(eng1,ok),           itt(eng2,ok),
oil_press(hyd_sys_1,ok),               oil_press(hyd_sys_2,ok),
airspeed(100), altitude(1000)],

 Oplist, Goalstate ). /* final state and how we got there */


             /* test2 dual eng failure */

  test2(Oplist,Goalstate) :-   problem_solver

       (    [    oil_press(eng1,none),   oil_press(eng2,none),
revs(rotor,low),         off(rain_rmv),         off(master_arm),
landing_zone(clear),     full(throttle1),     full(throttle2),
off(master_caution),     automatic(gov1),     automatic(gov2),
on(fuel1), on(fuel2), normal(fuel_press), torque(eng1,none),
torque(eng2,none),      torque(xmsn,low),      revs(ng1,none),
revs(ng2,none),         revs(nf1,none),        revs(nf2,none),
oil_temp(eng1,ok),      oil_temp(eng2,ok),      failed(gen1),
failed(gen2),       full(fuel),       off(scas),      on(ecu),
oil_temp(c_box,ok),  oil_press(c_box,ok), oil_temp(xmsn,ok),
oil_press(xmsn,ok),   ammeter(gen1,none),ammeter(gen2,none),
fuel_press(ok),         itt(eng1,none),        itt(eng2,none),
oil_press(hyd_sys_1,ok),               oil_press(hyd_sys_2,ok),
airspeed(100), altitude(1000), left_yaw ],

Oplist, Goalstate ).  /* final state and how we got there */


             /* test3 single eng failure */

  test3(Oplist,Goalstate) :-   problem_solver

([oil_press(eng1,none), oil_press(eng2,ok), revs(rotor,low),
off(rain_rmv),     off(master_arm),     landing_zone(clear),
```

```
  full(throttle1),    full(throttle2),    off(master_caution),
  automatic(gov1),   automatic(gov2),   on(fuel1),  on(fuel2),
  normal(fuel_press),    torque(eng1,none),   torque(eng2,ok),
  torque(xmsn,low),        revs(ng1,none),        revs(ng2,ok),
  revs(nf1,none),        revs(nf2,ok),        oil_temp(eng1,ok),
  oil_temp(eng2,ok),   failed(gen1),   full(fuel),  off(scas),
  on(ecu),        oil_temp(c_box,ok),       oil_press(c_box,ok),
  oil_temp(xmsn,ok),  oil_press(xmsn,ok),  ammeter(gen1,none),
  ammeter(gen2,ok),        fuel_press(ok),        itt(eng1,none),
  itt(eng2,ok),                         oil_press(hyd_sys_1,ok),
  oil_press(hyd_sys_2,ok),    airspeed(100),   altitude(1000),
  left_yaw ],

Oplist, Goalstate ).  /* final state and how we got there */

     /* test4 single eng1 failure + nf2 gov failure */

   test4(Oplist,Goalstate) :-   problem_solver

  ([oil_press(eng1,none), oil_press(eng2,ok), revs(rotor,low),
  off(rain_rmv),     off(master_arm),     landing_zone(clear),
  full(throttle1),    full(throttle2),    off(master_caution),
  automatic(gov1),   automatic(gov2),   on(fuel1),  on(fuel2),
  normal(fuel_press), torque(eng1,none), torque(eng2,erratic),
  torque(xmsn,low),    torque(xmsn,erratic),   revs(ng1,none),
  revs(ng2,erratic),     revs(nf1,none),     revs(nf2,erratic),
  oil_temp(eng1,ok),       oil_temp(eng2,ok),      failed(gen1),
  full(fuel),    off(scas),    on(ecu),    oil_temp(c_box,ok),
  oil_press(c_box,ok),  oil_temp(xmsn,ok), oil_press(xmsn,ok),
  ammeter(gen1,none),   ammeter(gen2,ok),  fuel_press(  ok  ),
  itt(eng1,none),  itt(eng2,erratic), oil_press(hyd_sys_1,ok),
  oil_press(hyd_sys_2,ok),    airspeed(100),   altitude(1000),
  left_yaw ],

Oplist, Goalstate ).  /* final state and how we got there */
```

LIST OF REFERENCES

1. Stegner, V. F., and Campbell, H. W., Artificial Intelligence for Aircrew Assistance Aeronautical Systems Division, Wright-Patterson AFB, OH May 1985.

2. "Technical Survey: Artificial Intelligence", Aviation Week and Space Technology, pp. 40-92, February 17 1986.

3. Decker, W. L., Application Of Artificial Intelligence To Improve Aircraft Survivability, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1985.

4. Chow, E. Y. H., Decision Support Systems For The Commercial Aviation Industry, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1981.

5. Winston, P. H. Artificial Intelligence, Addison Wesley, 1984.

6. Turbo Prolog (tm) Version 1.0 Copyright 1986 ,Borland International, Scotts Valley, Ca.

7. "DARPA Envisions New Generation of Machine Intelligence Technology",Aviation Week and Space Technology, Apr 22, 1985.

8. "Lockheed, McDonnell Win Pilot's Associate AI Contract", Aviation Week and Space Technology, Feb. 10 1986.

9. "Expert Systems Research Focuses on Combat Emergency Procedures", Aviation Week and Space Technology, Oct. 28, 1985.

10. Cohen P. R., and Feigenbaum E. A., The Handbook of Artificial Intelligence, V. 3, William Kaufmann, Inc., 1982.

11. Rowe, N. C. AI Through Prolog, Prentice-Hall 1987

12. "Prolog A Language for Artificial Intelligence",PC Magazine, October 14, 1986.

13. Shafer, D. Turbo Prolog Primer, Howard W. Sams and Co., 1986.

BIBLIOGRAPHY

Naval Air Systems Command NATOPS Flight Manual Navy Model AH-1T(TOW) 01-H1AAB-1, 1 August 1980.

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 2000 N. Beauregard Street Alexandria. VA 22311	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Mr. Russell Davis HQ, USACDEC Attention: ATEC-IM Fort Ord, CA 92152	2
Professor Neil C. Rowe, Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, CA 93943	10
Professor Robert B. McGhee, Code 52Mz Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2

Porter
A decision support system for the diagnosis of aircraft emergencies.